
بسم الله الرحمن الرحيم

۳-۶) تابع listen()

این تابع فقط در برنامه سرویس دهنده معنا می‌یابد و در یک عبارت ساده اعلام به سیستم عامل برای پذیرش تقاضاهای ارتباط TCP است. به عبارت بهتر توسط این تابع به سیستم عامل اعلام می‌کنید که از این لحظه به بعد (یعنی زمان اجرای تابع) تقاضاهای ارتباط TCP ماشینهای راه دور با شماره پورت مورد نظرتان را به صف کرده و منتظر نگه دارد.

با توجه به آنکه ممکن است پس از راه اندازی برنامه سرویس‌دهنده ، در لحظاتی چندین پروسه متفاوت بطور همزمان تقاضای برقراری ارتباط TCP به یک آدرس پورت بدهند بنابراین سیستم عامل باید بداند که حداکثر چند تا از آنها را بپذیرد و ارتباط آنها را به روش دست تکانی سه مرحله ای برقرار نموده و آنها را در صف سرویس دهی قرار بدهد. توسط تابع listen() باید به سیستم عامل اعلام شود که حداکثر تعداد ارتباطات فعال و باز روی یک شماره پورت خاص چند تا باشد. فرم کلی تابع بصورت زیر است:

```
int listen(int sockfd, int backlog);
```

sockfd : همان مشخصه سوکت است که در ابتدا آنرا ایجاد کرده‌اید.
backlog : حداکثر تعداد ارتباطات معلّق و به صف شدهٔ منتظر. در بسیاری از سیستمها مقدار backlog به ۲۰ محدود شده است.

همانند توابع قبلی در صورت بروز خطا مقدار برگشتی این تابع ۱- خواهد بود و متغیر errno شماره خطای رخداده می‌باشد.

۴-۶) تابع accept()

این تابع اندکی مرموز به نظر می‌رسد و بایستی به مفهوم آن دقت شود :
پس از آنکه تابع listen() اجرا شد تقاضای ارتباط TCP پروسه های روی ماشینهای راه دور (درصورت وجود) پذیرفته ، به صف شده و معلق نگاه داشته میشود. وقتی که تابع accept() اجرا می‌شود در حقیقت برنامه شما از سیستم عامل تقاضا می‌کند که از بین تقاضاهای به صف شده یکی را انتخاب کرده و آنرا با مشخصات پروسه طرف مقابل تحویل برنامه بدهد. بنابراین برنامه باید از بین ارتباطات معلق یکی را به

حضور بطلبد تا عملیات لازم را انجام بدهد. بهمین دلیل سیستم عامل یک مشخصه سوکت جدید ایجاد کرده و آنرا به برنامه بر می‌گرداند. در اینجا شما یک سوکت جدید دارید. مشخصه سوکت اول که توسط تابع socket() بدست آمده و مشخصه سوکت دوم که با تابع accept() به برنامه شما برگشته است. تفاوت این دو سوکت در چیست؟

الف: از سوکت اول برای پذیرش یکی از ارتباطات معلق در دستور accept() استفاده می‌کنید. در حقیقت این سوکت مشخصه کل ارتباطات به صف شدهٔ منتظر میباشد.

ب: از سوکت دوم برای دریافت و ارسال اطلاعات روی یکی از ارتباطات معلق استفاده می‌کنید. این سوکت مشخصه یکی از ارتباطات به صف شده می‌باشد.

فرم کلی تابع به صورت زیر است :

```
#include <sys/socket.h>

int accept(int sockfd, void *addr, int *addrlen);
```

sockfd : مشخصه سوکت است که در ابتدا با تابع socket() بدست آمده است.

addr : اشاره‌گر به استراکچری است که شما آنرا بعنوان پارامتر به این تابع ارسال می‌کنید تا سیستم عامل پس از پذیرش یک ارتباط معلق آدرس پورت و آدرس IP طرف مقابل ارتباط را در آن به برنامه شما برگرداند. ساختار این استراکچر قبلاً معرفی شد.

addrlen : طول استراکچر addr بر حسب بایت

مقدار برگشتی این تابع یک مشخصه سوکت است که در روالهای بعدی مورد استفاده قرار می‌گیرد. اگر مقدار برگشتی (۱-) باشد خطائی رخداده است که شماره آن خطا در متغیر سراسری errno قابل بررسی است.

مثال ناتمام زیر برای روشن شدن کلیت کار بسیار سودمند خواهد بود:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#define MYPORT 3490    /* the port users will be connecting to */
#define BACKLOG 10     /* how many pending connections queue will hold */

main()
{
   int sockfd, new_fd;  /* listen on sock_fd, new connection on new_fd */
   struct sockaddr_in my_addr;    /* my address information */
   struct sockaddr_in their_addr; /* connector's address information */
   int sin_size;

   if ((sockfd = socket(AF_INET, SOCK_STREAM, 0))!= NULL){

   my_addr.sin_family = AF_INET;         /* host byte order */
   my_addr.sin_port = htons(MYPORT);     /* short, network byte order */
   my_addr.sin_addr.s_addr = INADDR_ANY; /* auto-fill with my IP */
   bzero(&(my_addr.sin_zero), 8);        /* zero the rest of the struct */

   if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr))!=-1){

   listen(sockfd, BACKLOG);

   sin_size = sizeof(struct sockaddr_in);
   new_fd = accept(sockfd, &their_addr, &sin_size);
   .
   .
   .
```

بـار دیگـر تـأکید می‌کنیـم کـه بـرای ارسـال یـا دریـــافت داده‌هـــا بایســـتی از ســـوکت جدیدی که مشخصه آن توســـط تـابع ()accept برمـی‌گـردد ، اسـتفاده کنیـد.

۶-۵) توابع ()send و ()recv

ایـن دو تـابع در برنامـه سـمت سـرویس دهنـده و برنامـه سـمت مشـتری قـابل اسـتفاده بوده و برای مبادله داده‌ها کاربرد دارند. فـرم کلـی دو تـابع بـه صـورت زیـر اسـت :

```
int send(int sockfd, const void *msg, int len, int flags);

int recv(int sockfd, void *buf, int len, unsigned int flags);
```

sockfd : مشخصه سـوکتی کـه از تـابع ()accept بدسـت آمـده اسـت.
msg : محلی در حافظه (مثـل آرایـه یـا اسـتراکچر) کـه داده‌هـای ارسـالی از آنجـا اسـتخراج شده و درون فیلد داده[1] از یـک بسـته TCP قـرار گرفتـه و ارسـال می‌شـوند.
len : طول داده‌های ارسالی یـا دریـافتی بـر حسـب بـایت
flag : برای پرهـیز از پیچیـدگـی بحـث در ایـن مـورد توضیـح نمی‌دهیـم. فقـط در آن صفـر بگذاریـد.

buf : این پارامتر در تابع recv() آدرس محلی در حافظه است که داده‌های دریافتی در آنجا قرار گرفته و به برنامه باز گردانده می‌شود.

مقدار برگشتی این دو تابع در صورت بروز هر گونه خطا ۱- خواهد بود ولی در صورت برگشت یک عدد مثبت ، تعداد بایتهای ارسالی یا دریافتی را بر حسب بایت مشخص می‌کند. دقت کنید که ممکن است تعداد بایتهای ارسالی یا دریافتی با تعدادی که در متغیر len تقاضا داده‌اید یکسان نباشد. بعنوان مثال فرض کنید شما در متغیر len مقدار ۱۰۰۰ قرار داده‌اید ولی مقداری که تابع برگردانده است ۲۰۰ باشد. در این صورت ۸۰۰ بایت از کل داده‌های ارسالی (یا دریافتی) باقی مانده است که برنامه شما باید تکلیف آنها را مشخص کند.

---

[۱] Payload

توصیه : در هر مرحله سعی کنید حجم داده‌هائی که توسط تابع send() ارسال می‌کنید حول و حوش یک کیلو بایت باشد.

نکته : توابع send() , recv() فقط برای ارسال و دریافت روی سوکتهای نوع استریم کاربرد دارد ولی اگر می‌خواهید به روش UDP و با سوکتهای دیتاگرام داده‌هایتان را ارسال کنید اندکی صبر کنید؛

۶-۶) توابع close() و shutdown()

تا زمانی که نیاز داشتید میتوانید یک ارتباط را باز نگه‌داشته و داده ارسال یا دریافت نمائید ولیکن همانند فایلها هرگاه نیازتان برطرف شد باید ارتباط را ببندید.

فرم کلی تابع close() بصورت زیر است :

```
close(int sockfd);
```

sockfd : مشخصه سوکت مورد نظر. این سوکت همان مشخصه ای است که تابع accept() برگردانده است. دقت کنید که اگر sockfd مشخصه ای باشد که توسط تابع socket() برگشته است تمام ارتباطات معلق و منتظر نیز بسته خواهد شد.

ارتباطی که توسط تابع close() بسته می‌شود دیگر برای ارسال و دریافت قابل استفاده نخواهد بود.

هر گاه سوکتی را ببندید در حقیقت یکی از ارتباطات TCP را بسته اید و سیستم عامل می‌تواند بجای آن تقاضای ارتباط دیگری را قبول کرده ، برای پردازش به صف ارتباطات معلق اضافه کند.

راه دیگر بستن یک سوکت تابع shutdown() می‌باشد که فرم کلی آن بصورت زیر است :

```
int shutdown(int sockfd, int how);
```

sockfd : مشخصه سوکت مورد نظر

how : روش بستن سوکت که یکی از سه مقدار زیر را می‌پذیرد:

- مقدار صفر: دریافت داده را غیر ممکن می‌سازد ولی سوکت برای ارسال داده ، همچنان باز است. سیستم عامل بافر ورودی[1] مربوط به آن سوکت را آزاد می‌کند.
- مقدار ۱ : ارسال داده را غیر ممکن می‌سازد در حالی که سوکت برای دریافت داده‌ها همچنان باز است. سیستم عامل بافر خروجی[2] مربوط به آن سوکت را آزاد می‌کند.

- مقدار ۲: ارسال و دریافت را غیر ممکن کرده سوکت کاملاً بسته می‌شود. این حالت دقیقاً همانند تابع close() عمل می‌نماید.

همانند توابع قبلی در صورت بروز خطا مقدار برگشتی این توابع ۱- خواهد بود و متغیر سراسری errno شماره خطا را برای پردازش مشخص می‌کند.

۷) توابع مورد استفاده در برنامه مشتری (مبتنی بر پروتکل TCP)

تا اینجا توابعی که معرفی شدند توابع پایه‌ای بودند که در سمت سرویس دهنده به نحوی استفاده می‌شوند. حال باید ببینیم در سمت مشتری چه توابعی مورد استفاده قرار می‌گیرند:

الف : ابتدا دقیقاً مانند برنامه سرویس دهنده یک سوکت بوجود بیاورید. برای اینکار از تابع socket() که در بخش قبلی معرفی شد استفاده کنید. تا اینجا هیچ تفاوتی برای بکارگیری این تابع در سمت سرویس دهنده و سمت مشتری وجود ندارد.

ب : در هنگام نیاز مستقیماً تقاضای برقراری ارتباط را به سمت سرویس دهنده بفرستید و آنقدر منتظر شوید تا این تقاضا پذیرفته شود. این عمل توسط تابع connect() انجام می‌شود که در ادامه توضیح داده خواهد شد.

ج – از توابع send() و recv() برای ارسال و دریافت داده‌ها استفاده کنید.

د– نهایتاً ارتباط ایجاد شده را توسط تابع close() یا shutdown() ببندید.

---

[۱] Inbound buffer
[۲] Outbound buffer

۱-۷) تابع connect()

برای برقراری ارتباط با یک سرویس دهنده از تابع connect() استفاده می‌شود و در صورتی که برنامه سرویس دهنده روی ماشین مورد نظر اجرا شده باشد و توابع listen() و accept() در برنامه فراخوانی شده باشند آنگاه نتیجه تابع connect() موفقیت آمیز خواهد بود.

فرم کلی تابع connect() به صورت زیر است :

```
#include <sys/types.h>
#include <sys/socket.h>

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);
```

sockfd : مشخصه سوکتی است که با فراخوانی تابع socket() بدست آمده است.

serv_addr : استراکچری از نوع sockaddr است که قبلاً معرفی شد. در این استراکچر آدرس IP ماشین مقصد و آدرس پورت برنامه مقصد تعیین خواهد شد.

addrlen : اندازه استراکچر قبلی را بر حسب بایت معرفی می‌کند و می‌توان براحتی در این پارامترمقدار sizeof(struct sockaddr) قرار داد.

به این نکته دقت کنید که شما آدرس پورت خودتان را تنظیم نمی‌کنید بلکه سیستم عامل بطور خودکار یک شماره پورت تصادفی برای شما انتخاب می‌کند و مقدار این شماره برای برنامه سمت مشتری اصلاً مهم نیست چرا که وقتی شما به یک سرویس دهنده متصل می‌شوید و سرویس دهنده این تقاضا را می‌پذیرد پاسخ سرویس دهنده به همان آدرس پورتی خواهد بود که سیستم عامل برای سوکت انتخاب کرده است. در حقیقت برنامه شما بعنوان شروع کنندهٔ ارتباط ، آدرس پورت خود را نیز به طرف مقابل اعلام می‌کند. در مقابل آدرس پورت برنامه سرویس دهنده قطعاً باید ثابت و مشخص باشد تا برنامه مشتریها بتوانند ارتباط را شروع نمایند.
در صورت عدم موفقیت در برقراری یک ارتباط TCP مقدار برگشتی این تابع ۱- خواهد بود و متغیر errno شماره خطای رخداده می‌باشد.

مثال ناتمام زیر تا حدودی این دیدگاه را به شما عرضه می‌کند:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#define DEST_IP   "132.241.5.10"
#define DEST_PORT 23

main()    {

      int sockfd;
      struct sockaddr_in dest_addr;   /* will hold the destination addr */

      if (( sockfd = socket(AF_INET, SOCK_STREAM, 0))!=NULL) {

       dest_addr.sin_family = AF_INET;             /* host byte order */
       dest_addr.sin_port = htons(DEST_PORT);  /* short, network byte order */
       dest_addr.sin_addr.s_addr = inet_addr(DEST_IP);
       bzero(&(dest_addr.sin_zero), 8);              /* zero the rest of the struct */

     if ((connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)))!=-1) {
       .
       .
       .
```

۸) ارسال و دریافت به روش UDP با سوکتهای دیتاگرام

توابع ارسال ، دریافت و پذیرش برای سوکتهای نوع استریم کاربرد دارد. حال باید دید که به چه صورت می‌توان ارسال و دریافت را به روش UDP روی سوکتهای نوع دیتاگرام انجام داد.

- **برنامه سمت سرویس دهنده**

الف : یک سوکت از نوع دیتاگرام ایجاد کنید. این کار با فراخوانی تابع socket() با پارامتر SOCK_DGRAM انجام می‌شود.

ب : به سوکت ایجاد شده آدرس پورت مورد نظرتان را نسبت بدهید. (با تابع bind())

ج : بدون هیچ کار اضافی میتوانید منتظر دریافت داده‌ها بشوید. (تا موقعی که داده‌ای دریافت نشود ارسال معنی نمی‌دهد.) وقتی داده‌ای دریافت و پردازش شد آدرس برنامه مبدا (آدرس IP و پورت) مشخص شده و ارسال امکان پذیر خواهد بود.

ارسال و دریافت روی سوکتهای نوع دیتاگرام بوسیله توابع recvfrom() و sendto() انجام می‌شود.

د: نهایتاً سوکت ایجاد شده را ببندید.

- **برنامه سمت مشـــتری**

الـف: یـک سـوکت از نـوع دیتـــاگـــرام ایجـــاد کنیـــد. (بـــا تـــابع ()socket و پـــارامتر SOCK_DGRAM)

ب: هر گاه نیـاز شـد بـدون هیـچ کـار اضـافی داده‌هایتـان را بـه سـمت سـرویس دهنـده ارسـال نمـایید. تـا وقتـی کـه بـه سـمت سـرویس دهنـده ارسـالی نداشـته باشـید، دریـــافت داده‌ها معنا نمی‌دهد چرا کـــه شـما بـرای سـرویس دهنـده شـناخته شـده نیسـتید مگـر اینکـه داده‌ای را ارسال نمائید. ارسال و دریافت را تـا زمـانی کـه نیـاز اسـت انجـام بدهیـد.

ج: سوکت ایجـاد شـده را ببندیـد.

فرم کلی تابع ارسال داده مبتنی بر ســوکتهای دیتـاگـرام بصـورت زیـر اسـت:

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to,
    int tolen);
```

sockfd : مشخصه سوکت دیتاگرام کـــه بـا تـابع ()socket بوجـود آمـده اسـت.

msg : آدرس محـل قـرارگرفتـن پیـام در حافظـه کـه داده‌هـــای ارســـالی بایســـتی از آنجـــا استخراج شده و درون یـک بسـته UDP قـرار گرفتـه و ارسـال شـود.

len : طول پیام ارسالی بـــر حسـب بـایت

flags : برای پرهیز از پیچیدگی بحث فعلاً آنرا صفر در نظر بگیرید.
to : استراکچری از نوع sockaddr که قبلاً ساختار آنرا مشخص کردیم. در این استراکچر باید آدرس IP مربوط به ماشین مقصد و همچنین شماره پورت سرویس دهنده تنظیم شود.
tolen : طول استراکچرsockaddr است که به سادگی می‌توانید آنرا به مقدار sizeof(struct sockaddr) تنظیم نمایید.

مقدار برگشتی این تابع همانند تابع send() تعداد بایتی است که سیستم عامل موفق به ارسال آن شده است. دقت کنید که اگر مقدار برگشتی (1-) باشد خطائی بروز کرده که می‌توانید شماره خطا را در متغیر سراسری errno بررسی نمائید. باز هم تکرار می‌کنیم دلیلی ندارد تعداد بایتی که تقاضای ارسال آنها را داده‌اید با تعداد بایتی که ارسال شده یکی باشد. بنابراین حتماً این مورد را در برنامه خود بررسی کرده و همچنین تقاضاهای ارسال در هر مرحله را نزدیک یک کیلو بایت در نظر بگیرید.

فرم کلی تابع دریافت داده مبتنی بر سوکتهای دیتاگرام بصورت زیر است:

```
int recvfrom(int sockfd, void *buf, int len, unsigned int flags,struct sockaddr *from,
        int  *fromlen);
```

sockfd : مشخصه سوکت دیتاگرام که با تابع socket() بوجود آمده است.
buf : آدرس محلی از حافظه که سیستم عامل داده‌های دریافتی را در آن محل قرار خواهد داد.
len : طول پیامی که باید دریافت شود (بر حسب بایت)
from : استراکچری است از نوع sockaddr که قبلاً ساختار آن بررسی شد و سیستم عامل آنرا با مشخصات آدرس IP و آدرس پورت برنامه مبداء تنظیم و به برنامه شما برمی‌گرداند.
flag : آنرا به صفر تنظیم کنید.
len : طول استراکچری است که سیستم عامل آنرا برگردانده است.

مقدار برگشتی این تابع نیز تعداد بایتی است که دریافت شده است. این پارامتر برای پردازش داده‌های دریافتی اهمیت حیاتی دارد.

۹) توابع مفید در برنامه نویسی شبکه

بغیر از توابع سیستمی معرفی شده توابع دیگری هم هستند که برای برنامه نویسی شبکه بسیار مفید و کارآمد هستند. در ادامه برخی از مهمترین آنها را تشریح خواهیم کرد:

۹-۱) تابع getpeername()

```
#include <sys/socket.h>

int getpeername(int sockfd, struct sockaddr *addr, int *addrlen);
```

با استفاده از این تابع می‌توانید هویت طرف مقابل ، شامل آدرس IP و آدرس پورت پروسهٔ طرف مقابل ارتباط را استخراج نمائید. پارامترهای این تابع بصورت ذیل تعریف شده است :

sockfd : مشخصه سوکت مورد نظر

addr : استراکچری است از نوع sockaddr که قبلاً ساختار آن تعریف شده است. این استراکچر توسط سیستم عامل با آدرس IP و آدرس پورت طرف مقابل پر شده است.

addrlen : طول استراکچرsockaddr

در صورت عدم موفقیت تابع فوق مقدار برگشتی (۱-) خواهد بود و در متغیر سراسری errno شماره خطا برای بررسی نوع خطا تنظیم خواهد شد.

نکته ای که ممکن است برنامه نویس فراموش کند آن است که ترتیب آدرس IP و آدرس پورت بصورت BE است و اگر ماشین شما از نوع LE است باید حتماً آنرا تبدیل کنید.

۲-۹) تابع gethostname()

این تابع نام ماشینی را که برنامهٔ شما روی آن اجرا می‌شود ، بر خواهد گرداند. این نام یک رشته کاراکتری معادل با نام نمادین ماشین است نه آدرس IP آن (مثلاً www.ibm.com). فرم کلی تابع بصورت زیر است:

```
#include <unistd.h>

int gethostname(char *hostname, size_t size);
```

hostname : یک آرایه از کاراکترها (یا بعبارت بهتر یک رشته کاراکتری) است که پس از بازگشت تابع نام ماشین در آنجا ذخیره خواهد شد.

size : طول رشته کاراکتری بر حسب کاراکتر

اگر مقدار برگشتی (۱-) باشد خطائی بروز کرده و مقدار errno همانند قبل شماره خطا را نگه می‌دارد ولی اگر تابع فوق موفق عمل کند مقدار برگشتی صفر خواهد بود.

۳-۹) بکارگیری سیستم DNS برای ترجمه آدرسهای حوزه

قبلاً در مورد سیستم DNS و طرز عملکرد آن بحث شد. در اینجا وقت آن فرا رسیده است که بتوانید در محیط برنامه نویسی تقاضای ترجمه نام حوزه[۱] یک سرویس دهنده را به این سیستم ارائه کرده و نتیجه را در برنامه خود استفاده نمائید.

---

[۱] Domain Name

مثالهای کوچکی که تا اینجا داشته ایم همگی برای برقراری یک ارتباط با ماشین خاص مستقیما از آدرس IP آن استفاده می‌کردند و لیکن فرض کنید که شما بخواهید برنامه ای بنویسید که کاربر بتواند آدرس نام حوزه یک سرویس دهنده را بعنوان آدرس مقصد وارد نماید. تابعی که در این مورد بکار می‌آید دارای فرم کلی زیر است:

```
#include <netdb.h>

struct hostent *gethostbyname(const char *name);
```

name : رشته کاراکتری نام حوزه یک سرویس دهنده

مقدار برگشتی تابع ، آدرس استراکچری است از نوع hostent که ساختار آن بصورت زیر تعریف شده است :

```
struct hostent {
      char   *h_name;
      char   **h_aliases;
      int    h_addrtype;
      int    h_length;
      char   **h_addr_list;
   };

#define h_addr h_addr_list[0]
```

hname : نام رسمی ماشین (برای شبکه اینترنت این رشته نام حوزه خواهد بود مثلاً (www.ibm.com

h_aliases : نام مستعار ماشین (این رشته با \0 ختم می‌شود)

h_addrtype : خانواده آدرس (همانگونه که اشاره شد در شبکه اینترنت این فیلد مقدار AF_INET خواهد داشت.)

h_length : طول آدرس بر حسب بایت

h_addr_list : یک رشته کاراکتری که در آن آدرس IP مربوط به ماشین سرویس دهنده قرار دارد. این رشته با \0 ختم می‌شود.

دقت کنید که در تابع بالا در صورت موفقیت آمیز بودن، یک اشاره گر به استراکچر بر می‌گرداند و در غیر اینصورت مقدار NULL برخواهد گشت و برخلاف

توابع قبلی متغیر errno تنظیم نخواهد شد و بجای آن متغیر سراسری herror که متغیری سیستمی است تنظیم می‌شود و در ضمن تابع سیستمی herror() برای کشف نوع خطا بکار گرفته می‌شود.

برای رفع ابهاماتی که در این تابع وجود دارد طرح یک مثال ضروری به نظر می‌رسد. به برنامه کوچک و اجرائی زیر دقت نمائید :

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>

int main(int argc, char *argv[])
{
```

```
    struct hostent *h;

    if (argc != 2) {  /* error check the command line */
       fprintf(stderr,"usage: getip address\n");
       exit(1);
    }

    if ((h=gethostbyname(argv[1])) == NULL) {  /* get the host info */
       herror("gethostbyname");
       exit(1);
    }

    printf("Host name  : %s\n", h-h_name);
    printf("IP Address : %s\n",inet_ntoa(*((struct in_addr *)h-h_addr)));

    return 0;
}
```

نام این برنامه getip است که یک آدرس نام حوزه را بعنوان ورودی دریافت کرده و نتیجه ترجمه آن را به آدرس IP و بقیه مشخصات را روی خروجی چاپ می‌کند.
نکات زیر درمورد برنامه بالا ارزش بازگوئی دارد:

الف: طریقه بکارگیری برنامه فوق بدینصورت است که نام برنامه را روی خط فرمان تایپ کرده و سپس در جلوی آن نام حوزه را با یک فاصله خالی نوشته و کلید Enter را فشار می‌دهید. مثال :

ب: آدرس IP معادل با آدرس نام حوزه در متغیر h_addr_list واقع است و هر چند که بصورت یک رشته است که با کد 0\ ختم می‌شود ولی برای شبکه اینترنت که آدرسهای IP فعلاً چهار بایتی هستند شما فقط به چهار بایت اول آن که بصورت BE ذخیره شده‌اند نیازمندید. در برنامه فوق برای تبدیل آدرس چهاربایتی به حالت رشته ای نقطه دار بفرم (مثلاً 213.190.140.187) ازتابع inet_ntoa() برای چاپ روی خروجی بهره گرفته شده است.

ج: عمل "تطبیق نوع" در تابع inet_ntoa() به آن دلیل بوده است که طبق تعریف اصلی متغیر h→h_addr بصورت رشته معمولی تعریف شده ولی در تابع inet_ntoa() آرگومان ورودی آن یک استراکچر از نوع in_addr است که در ابتدای فصل ساختار آن تعریف شد و چهاربایتی است. بنابراین مجبوریم با عمل "تطبیق نوع" سازگاری پارامتر ورودی را تضمین کنیم ولی در عمل اتفاق خاصی نمی‌افتد.